

Using the Model Paradigm for Real-Time Systems Development: ACCORD/UML

Sébastien Gérard, François Terrier and Yann Tanguy

(Sebastien.Gerard, Francois.Terrier, Yann.Tanguy)@cea.fr
CEA / DRT-LIST / DTISI / SLA / L-LSP
F-91191 Gif sur Yvette Cedex France
Phone: +33 1 69 08 58 24
Fax: +33 1 69 08 20 82

Abstract. Industrial competition is intensifying and products are becoming ever more complex and software-intensive. Engineers must therefore face the challenge of being both business experts and advanced software developers. This situation urgently requires new ways of thinking about software development, with the primary objective of minimizing development efforts wherever possible, so that engineers can focus on the added value of their products. The latest OMG initiative, called MDA -- for "Model Driven Architecture" -- advances the idea of using a combination of the model paradigm and various underlying technologies, to offer solutions to this challenge. For real-time domain applications, engineers may already be using model-orientated approaches ([1], [2], ...). However, these are not yet entirely satisfactory, since they still require advanced real-time development skills. This paper describes the ACCORD/UML approach that was originally designed to lighten the daily workload of automotive engineers by relieving them of need for real-time expertise.

1 Introduction

Faced with intensifying industrial competition, engineers are increasingly being asked to focus on the market research and business development sides of their activities. This means finding ways to differentiate their products, making them more and more innovative than those of their competitors. It also entails further reduction in the famous "time-to-market" and constantly greater product complexity! In parallel, products are becoming ever more software-intensive, and thus also require high-level skills in software engineering. Today's engineers must therefore face the challenge of being both business experts and advanced software developers.

The automotive domain, for example, is very representative of such a situation. It is an area where competition is relentless and engineers must be constantly "creative". Moreover, new vehicle functions are relying more and more on software capabilities requiring advanced software engineering know-how. Since such know-how evolves quickly, it is also becoming more difficult for engineers to find time to up-

date their software development skills. Functions such as ABS, navigation systems and engine control are illustrative of this state of affairs.

It is therefore urgent to define new techniques for software development that allow engineers to concentrate on their business approach. In response to such challenges, as stated in [3], one emerging solution could be to switch from code-oriented to model-oriented development of software intensive applications. In this respect, the model paradigm, where suitably supported by industrial toolkits, seems very promising as a means for helping engineers master the complexities of state-of-the-art applications.

OMG has performed a considerable amount of work around CORBA for defining standard middleware to manage the development problems associated with distributed systems in heterogeneous environments. Over the last five years, the OMG Unified Modeling Language has also become the “de facto” language for several methodologies ([4], [1], [5], [2], [6], etc.). Now that CORBA and UML have each developed well on its own, OMG is trying to assemble these two paradigms and has coined a new technology concept -- MDA ([7], [8] and [9]). But what actually lies behind this “revolutionary” concept?

First of all, MDA means “Model-Driven Architecture”, and it is always teamed with two other important words in this context: “PIM” and its friend “PSM”. The “P” in both these acronyms stands for platform and refers “to technological and engineering details that are irrelevant to the basic functionalities of a software component” [8]. A PIM, “Platform Independent Model”, is also a work tool in which engineers only define and manipulate elements that are fully independent of any software implementation technology. The obvious advantage of this feature is that it should be easier to port/transform a business model into different operational environments relying on different implementation technologies. Indeed, once such a high level model, in this case the PIM, has been constructed, it becomes a PSM, i.e. a “Platform Specific Model”. At this stage, implementation technologies are chosen and the PIM is “implemented” on or via a PSM. For example a PIM may be transformed either into a PSM1 using technologies such as C++ and VxWorks, or into another configuration, PSM2, based on Java and Windows NT.

The first significant result of the MDA paradigm for engineers is the possibility for them to build application models that can be conveniently ported to new, emerging technologies - implementation languages, middleware, etc.- with minimal effort and risk.

In the real-time application area, this model-oriented trend is also very active and promising. Currently, there are four main model-oriented industrial approaches supported by tools: UML-RT used with Rose-RT, ROPES with Rhapsody, ARTiSAN and UML/SDL with the Tau UML/SDL suite.

Within UML-RT, an application is seen as a set of entities called “capsules” which support logical concurrency. These capsules have a state machine as behavior specification and may exchange signals to communicate. Models built in this way are said to be executable, meaning that at any moment in design, it is possible to produce an executable application matching the UML model. In this case, the mapping is achieved via code generation.

For ROPES and ARTiSAN approaches, real-time application modeling is a 3-stage process: i) building a “functional” model with class and state diagrams; ii) building a specific tasking model with class diagrams containing only active objects (~execution tasks); iii) describing the mappings between the two models. The main drawback of this "family" of methods is that it requires advanced real-time development skills to build the tasking model and map it with the “functional” model. While there are some "shortcuts" available ([10]) to facilitate this activity, no transformation rules are provided as could be done within a fully MDA-based approach.

The approach proposed by Telelogic is based on the use of both UML and SDL languages. It consists of building UML models at the analysis stages using active objects as concurrency supports and SDL within design-time. Reference document [11] defines modeling rules for mapping a UML-oriented model into an SDL-oriented model. When SDL models are finished, the engineer may generate code to produce an executable application.

All these methodologies may be considered as MDA-based approaches for mainly two reasons. Firstly, they clearly promote the model paradigm to develop applications; and secondly, they provide code generation taking into account structural and behavior specifications for model mapping to implementation languages such as C, C++, JAVA,...

Nevertheless, they do not exploit all the potentialities of MDA. Their application models are often only PSM-like for "executable" reasons. For modeling purposes, the user is thus led very quickly to resort, for an executable model, to a programming language such as C++, . . . Although action semantics have been standardized by OMG [12], there are still only a few tools that have integrated this feature, which allows building of executable models independently of any programming language.

While these approaches are usually based on a several stage process, they do not provide the refinement mapping rules that could facilitate application development and, above all, be highly useful in promoting seamless development processes.

Finally, the existing UML-based methods for real-time applications still require considerable knowledge of real-time software technology (and the different programming models promoted by these tools) to develop real-time systems.

The subject of this paper is then to present an approach called ACCORD/UML¹ dedicated to real-time application development. The audience for such a methodology includes engineers who are not “software” experts. Its aim is to provide both a method and the underlying tools in abstract enough form to enable specification and prototyping of real-time systems by non real-time experts.

As depicted in Fig. 1, ACCORD/UML relies on a basic three stage software development cycle that enables analysis, prototyping and testing. Progression from one to another of these phases is achieved by a seamless and iterative process of refining

¹ Parts of this article were taken from research conducted by Sébastien Gérard as part of his PhD thesis, in collaboration with the French car maker PSA; said research is now being partially supported by an European project under the 5th FRDP (Framework Research and Development Program): AIT-WOODDES (<http://wooddes.intranet.gr/>).

UML models². For that purpose, the ACCORD/UML profile [13] defines specific packages of model mappings. This profile also contains all UML extension definitions introduced within the ACCORD/UML context to facilitate access by non-experts to real-time systems development.

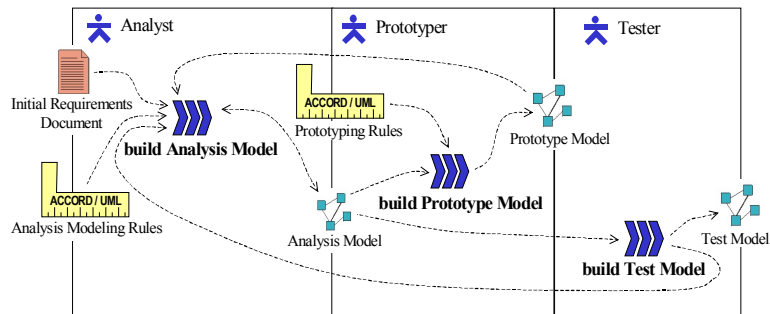


Fig. 1. Overview of the ACCORD/UML methodology³.

The global model of an ACCORD/UML-based application is in turn broken down into three sub-models: an Analysis Model, a Prototype Model and a Test Model.

Within MDA, one important technological issue concerns mappings which may be derived in different configurations such as PIM-to-PIM, PIM-to-PSM or PSM-to-PSM [8]. The purpose of this paper is to focus on mapping and to demonstrate that it is feasible and efficient to employ a full MDA-based methodology that uses model mappings intensively throughout the development process.

The rest of the paper is then structured as follows: First a description of the Analysis Model building phase. Since this first model is made up of two submodels, emphasis is placed on how to switch from one model to the other in a PIM-to-PIM mapping situation. This aspect is then described in the second section. The final section is devoted to conclusions on the efficiency of a full MDA-based approach and to potential future uses of the ACCORD/UML approach.

1 Analysis⁴ Model building phase

This phase serves to describe what the system is expected to do. All model elements involved here are derived from the problem domain and from the physical constraints imposed on the system (e.g. Automotive, Telecom, etc.). The product of this modelling is then used first as input for the prototyping phase (see section enti-

² In the rest of this paper, due to space limitation, we will focus on mappings defined within the analysis phase.

³ This figure uses the modeling artifacts defined in the OMG SPEM profile.

⁴ Use of the term “analysis” requires some clarification here. An “Analysis model” is a model that describes requirements and may also be called a “specification model”. It should not be confused with “model analysis” that may refer, for example, to model validation.

tled "Prototype Model building phase") to specify how the “what⁵” is to be performed. The Analysis Model is also used to build the test models that validate the application (see section entitled "Test Model building phase").

The Analysis Model is split into both the following submodels:

- Preliminary Analysis Model (PAM) – The PAM is intended to specify the overall functions of the application, in very general terms, as well as its interactions with the environment. Throughout this activity, the application is only considered from an external viewpoint, in other words as a “blackbox” interconnected with its environment;
- Detailed Analysis Modeling (DAM) – once the PAM is built, the user zooms in on the blackbox, and also builds the DAM. To do so, s/he describes as thoroughly and accurately as possible the structural, interactional and behavioral aspects of the application.

2.1 The “PAM” model

This first activity of the method thus consists of building a model called the Preliminary Analysis Model (in short PAM). This step plays a significant role in the project development cycle. It is the process activity during which product requirements (where they exist) can be reformatted as text and graphics that are easily accessible even to inexperienced users and also become formal⁶ specifications. Moreover, if not already the case, model building also offers the system analyst opportunity to become familiar with vocabulary and concepts specific to the application domain s/he is working for. As depicted in Fig. 2, a PAM is made up of a dictionary, a use case model and a high-level scenario model.

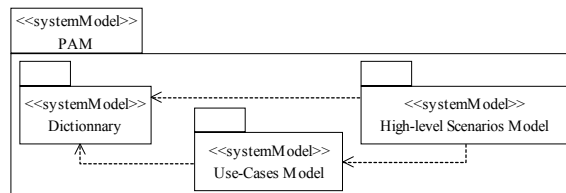


Fig. 2. A Preliminary Analysis Model.

2.2 The “DAM” model

The clear, unambiguous model of user needs that results from preliminary analysis still reflects primarily a functional view of the system. The aim of the subsequent

⁵ i.e. results of analysis that specify what the system is expected to do.

⁶ Formal means here that the resulting model will be interpreted in the same way by any user. For our purposes, formal specifications are not mandatory written in a mathematical-based language.

phase is also to build an object-oriented model based on user requirements expressed essentially as use cases and sequence diagrams. The second objective within DAM building is also to move from “a black-box” to a “white-box” perspective, with the latter detailing the content of the system. Throughout this stage, however, the engineer must remember that s/he is modeling the “what” and not the “how” of the system. To put it clearly, s/he is modeling requirements for implementation, not the implementation itself! For this reason, it is proposed to organize global model construction around three dependant partial, but complementary and consistent "sub-models" (Fig. 3):

(i) Structural Model – it defines the general architecture (topology) of the application in terms of classes and the relations between them. Its modeling function is limited to the "class" level of the application, i.e. to specifying both local class properties and those affecting the application as a whole (which means accounting for all necessary and possible interactions between classes). The structural model is described in particular by UML class diagrams;

(ii) Behavioral Model – it defines the behavior of classes involved in the application. It is likewise concerned with the "class" level only, i.e. with specifying class behavior. The behavioral model introduces two object views: that of the protocol, which specifies the global behavior (also known as the "life cycle") of the object; and the "triggering" view, which accounts for reactive behavior of objects such as reactions to received signals, periodic behavior, etc. The model may also describe the behavior of class operations;

(iii) Detailed Scenarios Model – it is concerned with application "instances" and defines message passing between these various instances for the purpose of performing a given task. The interaction model is specifically described by UML use case and sequence diagrams. It has an additional facet to enable specification of application start-up and initialization features of an application. The application installation is likewise specified via the interaction model, using a specific sequence diagram dedicated to this aspect.

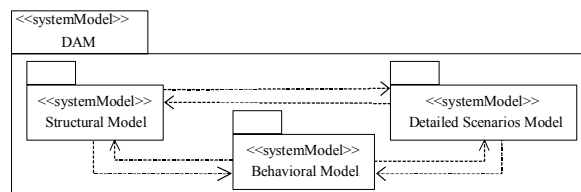


Fig. 3. A Detailed Analysis Model.

3 From PAM to DAM, the structural viewpoint

The structural aspect of a DAM is based on a layered model especially designed to clearly identify the connections between a system and its environment. This model thus enables structuring the system in such a way as to facilitate component devel-

opment by placing emphasis on specification of both "provided" and "required" interfaces. The following issues relating to component development are thus emphasized:

how a component is to be used, through clear definition of provided interface
what requirements are set for use of the component and how it is plugged into the environment needed to run it, through clear definition of a required interface?

To facilitate construction of an application in terms of reusable components, a generic software architecture is needed that emphasizes separation of the core of the system from its interface with the environment. This architecture is divided into the following five packages (Fig. 4).


The *ProvidedInterfaces* package describes active interaction points (i.e. where and how the environment "stimulates" the system). All the elements of this package play sensor roles in the system, i.e. they relay information from the environment to the system.

The lowermost package, called *RequiredInterfaces*, depicts the connection points at which the system interacts with its environment and models the environmental interface required for the system to operate. This package defines the specification that the environment needs to run the developed component. The elements in this package play an actuator role, i.e. they relay information from the system to its environment.

The middle layer describes the core of the system and is also called the Core package. It incorporates the actual inside of the application model. This package is a "zoom-in" on the single class introduced into the PAM model to represent the whole system.

For reasons of organization (to avoid confusing different concepts and preclude cyclic dependency among the packages), all of the signal definitions are grouped together in separate stereotyped «Signal» packages⁷ depending on their scope. As described in Fig. 4, an application may receive/send two kinds of signal: internal signals whose scope is the core package (these are considered as signals internal to the application); and external signals that may be shared with external systems/components located in the same namespace.

A last specific package can be added to this template of structure. It is shown on the left hand side of the figure. It is stereotyped «DomainTypes» and is introduced for the purpose of "collecting" particular business types. This is because, very often, the analyst is obliged to make design choices to model the requirement. For example, an automotive engineer may use a speed type. In this case, the first reflex of the analyst may be to say: "Let's have an integer or a float...". In our approach, s/he will then define a new type in this last package (e.g. "Speed" type) and does not need to make premature design choices.

⁷ A stereotyped «Signal» package is also depicted via the icon .

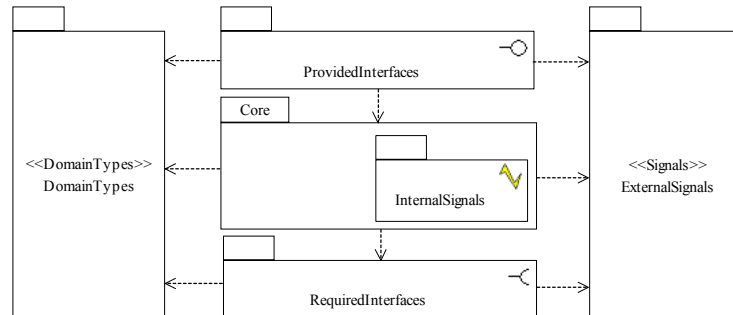


Fig. 4: Generic Structural Model of an ACCORD/UML-based Application.

Use of two specific interface packages and the choice of orientations for links between them are intended to ensure clear structuring and identification of dependencies between the developed system and its environment. Such design is necessary to facilitate system integration into an existing context and enhance the reusability of the application analysis models.

Regarding specific templates for structural models, the ACCORD/UML method defines, within its profile [13], the following modeling rules:

- A package stereotyped «Signals» must contain only signal-type elements.
- The package stereotyped «DomainTypes» must only contain model elements stereotyped «type».
- In the requirements analysis phase, stereotyped packages « ProvidedInterfaces » and « RequiredInterfaces » contain only interface classes.

During this step, some specific classes of the application may be identified automatically from the PAM. These are the System/Environment interface classes, also called interface classes. By applying the following modeling rules, the analyst populates both interface packages of the application in a systematic way:

- Each actor identified in the Use-Cases Model built during preliminary requirements analysis (i.e., the PAM) results in identification of a corresponding interface class in the model generated during detailed analysis (i.e., the DAM). All of the classes introduced in this way are interface classes⁸ (and are therefore either given an "interface" stereotype or depicted as circles with class name labels) and are assigned the same names as the actors they refine. Furthermore, if an actor is stereotyped as « active »⁹, the corresponding class is positioned in the provided interface package. If, on the other hand, it is stereotyped as « passive », it is included in the required interface package. An actor element in the PAM is linked via a dependency link stereotyped «refine» with its matching interface class in the

⁸ UML definition : "An interface is a named set of operations that characterize the behavior of an element."

⁹ Active and passive actors are determined thanks to the following modeling rule: "If an actor's role in a sequence diagram is only to output messages and receive responses to these messages, said actor is considered to be active and is therefore stereotyped as such. If, on the contrary, it serves only to receive messages sent by the system, it is said to be passive (even if it responds to the messages received) and is stereotyped accordingly."

DAM. Moreover, the direction of the dependency is from the interface class toward the actor.

The ACCORD/UML method is supported by a tool that allows implementation of the UML profile concept and, in particular, of modeling rules defined within a methodology. Mapping rules have been developed using the J language of Objecteering's UML Profile Builder [18]. This tool ensures extension of Objecteering capabilities to support a specific method, whether using standard UML extension mechanisms (stereotypes, tagged values, etc.) or adding behaviors using J language, e.g. by implementing model transformation rules as defined above.

4 Conclusion and outlook for the future

As stated in the introduction to this paper, one of the main issues for engineers in the coming years is a need for "double" expertise, in both business and software engineering, coupled with the increasing complexity and competitiveness in the business environment. The model paradigm appears as a possible solution for meeting this challenge. Throughout this paper, we have tried to demonstrate the main tenets of the AIT-WOODDES method with regard to MDA model progression. Two kinds of mapping are introduced: model refinement (e.g. PAM-to-DAM); and external model mapping (e.g. code and test generation). The main contribution of ACCORD/UML methodology is firstly its intensive use of model mappings at all stages in the development process, and secondly, the sets of rules defined within the underlying profile to facilitate model building/refinement at each of these stages.

The paper gave three examples to illustrate this point and demonstrate the efficiency of such a method. A PIM-to-PIM transformation was first depicted to demonstrate an internal model progression. This part of the paper described how to synthesize a draft of the DAM from the PAM.

For executable building, code generation rules are specified and implemented in an industrial UML-based tool. The generated code then allows the user to build a multi-tasking application without knowing anything about Real-Time Operating Systems.

Test model building relies on a formal tool, called AGATHA, that is able to compute all the symbolic execution paths of an application and generate an exhaustive set of behavioral test sequences. Model transformation is also intensively used. A file is first generated to the external tool, then the results are analyzed and retransformed into UML models, i.e., sequence diagrams.

The main point to be made here about model mapping rules is that up until now, these rules were defined informally (in natural language) and implemented with the proprietary language of an industrial UML-based tool that also depends on the tool support. The main requirement for incoming work will be to formalize these mapping rules within the ACCORD/UML profile so that they can be imported into any tool capable of accommodating them and that the methodology proposed here can be easily integrated into another UML-based tool that supports/imports UML profiles.

References

- [0] K.-D. Vöhringer, "Software and the Future of the Automotive Industry," in Proc. 2nd ITEA Symposium, Berlin, Germany, October 12th, 2001. <http://www.itea-office.org/>
- [1] B. Selic and J. Rumbaugh, "Using UML for Modeling Complex Real-Time Systems," ObjecTime Limited 98.
- [2] B. P. Douglass, "Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns," Addison Wesley, 99.
- [3] J. Bézivin, "From Object Composition to Model Transformation with the MDA," in Proc. of TOOLS USA, Santa Barbara, August 2001.
- [4] A. Lanusse, S. Gérard, and F. Terrier, "Real-Time Modeling with UML: The ACCORD Approach," in Proc. "UML98": Beyond the Notation, Mulhouse, France, 98.
- [5] F. D'Souza and A. Wills, "Objects, Components, and Frameworks with UML: the CATALYSIS Approach," vol. 1, 1999.
- [6] S. Gérard, P. Petterson, B. Josko, "Methodology for developing real time embedded systems," IST-1999-10069, 2002.
- [7] R. Soley and t. O. S. S. Group, "Model Driven Architecture (Draft 3.2)," OMG, White paper November 27, 2000 2000.
- [8] J. Miller and J. Mukerji, "Model Driven Architecture (MDA)," OMG, Specification July 9, 2001 2001.
- [9] J. Siegel and t. O. S. S. Group, "Developing in OMG's Model-Driven Architecture," OMG, White paper Revision 2.6, November, 2001.
- [10] M. Awad, J. Kuusela, and J. Ziegler, "Object-Oriented Technology for Real-Time Systems: A Practical Approach Using OMT and Fusion," Upper Saddle River, NJ 07458, USA: Prentice Hall, 96.
- [11] ITU-T, "Recommendation Z.109: Languages for telecommunications applications - SDL combined with UML," ITU-T, Geneva November 99 99.
- [12] OMG, "UML 1.4 with Action Semantics," OMG ptc/02-01-09, 2002.
- [13] S. Gérard, "The ACCORD/UML profile," CEA-LIST, Gif sur Yvette, Internal report 2002.
- [14] D. Lugato, N. Rapin, and J.-P. Gallois, "Verification and tests generation for SDL industrial specifications with the AGATHA," in Proc. of Workshop on Real-Time Tools, CONCUR'01, 2001.
- [15] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorenzen, "Object-Oriented Modelling and Design," Prentice Hall, 91.
- [16] I. Jacobson, M. Christerson, P. Jonson, and G. Övergaard, "Object-Oriented Software Engineering: A Use Case Driven Approach," 1992.
- [17] M. Broy, "Requirements Engineering for Embedded Systems," in proc. of Fem-Sys'97, 1997.
- [18] P. Desfray, "Profiles UML et langage J : Contrôlez totalement le développement d'applications avec UML," Softeam, Paris, white paper, 1999.